ECE 375 LAB 8

Remotely Operated Vehicle (USART)

Lab Time: Friday 4-6

Faaiq G. Waqar

Jordan T. Brown

Introduction

Lab 8 introduces the concept of communication between 2 microcontrollers through the USART protocol. During this lab we learned how to configure the USART registers on the atmega128 to enable the device to receive or transmit through IR signals. This allowed us to program multiple boards to act as bump bots that would receive from one transmitting remote. The remote will send commands to one bot that will manipulate with 5 commands that change the direction it drives. The final command sent to this bot will freeze all bots in its vicinity for 5 seconds. To achieve this result 2 separate programs are required for each type of bot, one for the transmitting remote and the other for the bump bots. The bots will still retain their bump bot behavior despite having the added capability to be frozen by the remote.

Program Overview

This program has two main portions, the transmission portion taking less code to complete than the receiver. We can break these up unto initialization, the most universal portion of the program, the main routines, where we using polling or interrupts to wait to call subroutines, send data through the receiver, receive data from the receiver, and process portions, including the freeze tag functionality. All discussed below.

Initialization Routine

The initialization routine is comprised of three main portions. The first is initialization of the I/O Ports, the next is the configuration of USART control registers A,B and C, and finally we will configure interrupts. For the transmission routine, the IO registers will enable the lights on the board for output using DDRB, this way we can track subroutine calls as the program moves along. We also enable the use of PORTD buttons set with the pull up resistor so that we may check for button presses, which will be used to send USART transmissions depending on the button press. For the USART control during transmission, we enable a 2400 baud rate using a double data rate by loading 832 into the UBRR high and low, and giving control register a value of 1 at the U2X bit to enable double data rate. Control B is used to enable the transmission, and the disable of the receiver. Finally in control C, we will enable 8 bit transmission with 2 enabled stop bits. For the receiver portion of this program, we will only change what we place into control register B, which will not receive the transmission bit, but rather an enable on the receiver and receiver enable bits to be set, so we may set it up for operation.

Main Routine

Main routine in both cases is very simple. In the case of the transmitter, it is a polling method that takes in the value from the buttons on PIND. The values are and with immediate to a binary that eliminates check on the Tx and Rx pins, and then checks for possible button presses, using a compare to expected binaries. These branches will go to routines that first send the address, wait for UDR to be cleared, send the respective values and then return. For the receiver, PORTB is being continuously loaded with the value of a register notated cpr used to load the current state, until an interrupt is received.

Send Data Routine

Sending data happens in one of the six instances, with the only difference being the code sent. These are branched to by the polling method mentioned in the main portion of the program. First, we check that the register has been cleared ahead of time, and wait until this is true. Then, we take the address of the receiver we would like to contact, and place this through a multi-purpose register to UDR, and via enabled transmission, send over USART.

The wait process for a clear of UDR is repeated, and then the specific command code is transmitted to the robot. After this, we return to the main portion of the program.

Receive Data Routine

Receiving the data is a little bit trickier. First, we will be reading from UDR to see what we get as the initial command. We know this can be one of two options, we are either receiving the freeze signal, in which case we will freeze the bot and increment the counter for freeze, or we will get the address. We should check the address to see if it meets out bot address, and continue to read the next portion only if its a match. If it is, we will read UDR again, and use the value to jump to a subroutine that loads the register cpr to the matching execution, then we clear the interrupt flags, and move back to the main program.

Freeze Command Routine

Freezing is sent normally through the transmitter, but handled differently than other functions in the receiver. How this works is that once a freeze command is received in the receiver, the receiver will turn off its receiving capabilities and turn on its transmission, and load the freeze command to the UDR, and send to other bots. This will allow us to send this signal to all bots around. A brief wait function is also used with receiver disable to make sure the bot does not freeze itself.

Additional Questions

No additional questions for this lab assignment

Difficulties

We had some trouble configuring all of the different bits for the 3 USART registers and it took many attempts to get the receiver functioning correctly, which we tested with a board the TA flashed with the receiver. Then once we started the receiver code we had trouble making progress and locating the source of our error. Our board would only receive a single command from one button and then stop working. Finally we realized that nothing was being loaded into USARTC and quickly complete the receiver afterwards.

Conclusion

This lab was by far the most challenging lab yet, but gave us lots of experience with many important skills. USART is still a very common protocol and has many real world applications as demonstrated in the lab slides. Implementing the freeze command was the most challenging part and tuning it to work just right also took a lot of time. Getting this amount of experience will make us much better assembly programmers and has greatly increased our understanding of how different communication protocols between devices work. This lab not only required knowledge of USART but also tested our skills using polling and interrupts.

Source Code - Transmitter

```
;***********************************************************

;*

;*      Faaiq_Waqar_and_Jordan_Brown_lab8_Tx_sourcecode.asm
```

```
;*
;*      This is the USART Reciever
;*
;*      This is the TRANSMIT skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*       Author: Faaiq Waqar & Jordan Brown
;*         Date: December 5th 2019
;*
;***********************************************************


.include "m128def.inc"                      ; Include definition file


;***********************************************************
;*      Internal Register Definitions and Constants
;***********************************************************
.def    mpr = r16                           ; Multi-Purpose Register
.def    spr     = r17                       ; Secondary-Purpose Register
.def    cpr = r18                           ; Comparative-Purpose Register


.equ    EngEnR = 4                          ; Right Engine Enable Bit
.equ    EngEnL = 7                          ; Left Engine Enable Bit
.equ    EngDirR = 5                         ; Right Engine Direction Bit
.equ    EngDirL = 6                         ; Left Engine Direction Bit
; Use these action codes between the remote and robot
; MSB = 1 thus:
; control signals are shifted right by one and ORed with 0b10000000 = $80
.equ    MovFwd =  ($80|1<<(EngDirR-1)|1<<(EngDirL-1))      ;0b10110000 Move Forward Action Code
```

```
.equ    MovBck =  ($80|$00)                                                    ;0b10000000
Move Backward Action Code

.equ    TurnR =   ($80|1<<(EngDirL-1))                                         ;0b10100000      Turn
Right Action Code

.equ    TurnL =   ($80|1<<(EngDirR-1))                                         ;0b10010000  Turn Left
Action Code

.equ    Halt =    ($80|1<<(EngEnR-1)|1<<(EngEnL-1))       ;0b11001000 Halt Action Code

.equ    Freeze =  0b11111000


;***********************************************************

;*      Start of Code Segment

;***********************************************************

.cseg                                                  ; Beginning of code segment


;***********************************************************

;*      Interrupt Vectors

;***********************************************************

.org    $0000                               ; Beginning of IVs

                rjmp    INIT                ; Reset interrupt


.org    $0046                               ; End of Interrupt Vectors


;***********************************************************

;*      Program Initialization

;***********************************************************

INIT:

        ;Stack Pointer

        ldi             mpr, high(RAMEND)

        out             SPH, mpr

        ldi             spr, low(RAMEND)

        out             SPL, spr
```

```
	;I/O Ports


	ldi		mpr, 0b11111111
	out		DDRB, mpr


	ldi		mpr, (1<<3)		; Set Port B Data Direction Register
	out		DDRD, mpr					; for output
	ldi		mpr, (1<<0|1<<1|1<<4|1<<5|1<<6|1<<7)	;  Initialize  Port  D  Data
Register
	out		PORTD, mpr					;  so  all  Port  D
inputs are Tri-State
	;USART1
		;Set baudrate at 2400bps
	ldi		mpr, high(832) ; Do We Round up or Down?
	sts		UBRR1H, mpr
	ldi		spr, low(832) ; Same Question applies here
	sts		UBRR1L, spr
		;Enable transmitter
	ldi		mpr, (1<<U2X1) ; Set Double Data Rate for Transmission
	sts		UCSR1A, mpr		; Load to Control Register A
	ldi		mpr, (1<<TXEN1|0<<UCSZ12)	;Enable the transmitter
	sts		UCSR1B, mpr				;Load to COntrol Register B
	ldi		mpr,	(0<<UMSEL1|0<<UPM11|0<<UPM10|1<<USBS1|1<<UCSZ11|1<<UCSZ10)	;
Enable8 bits and 2 stop
	sts		UCSR1C, mpr				; Load to control register C
	;Set frame format: 8 data bits, 2 stop bits


	;Other


;***********************************************************
;*	Main Program
```

```
;***********************************************************

MAIN:

              in            cpr, PIND                   ;  Use  polling  method  and  take  in
Buton input

              andi    cpr, 0b11110011             ; And with this binary to eliminate issues
with Tx/Rx Bits

              cpi           cpr, 0b01110011             ; Forward Command Check

              breq    USART_FWD

              cpi           cpr, 0b10110011             ; Backwars Command Check

              breq    USART_BCK

              cpi           cpr, 0b11010011             ; Right Command Check

              breq    USART_RGT

              cpi           cpr, 0b11100011             ; Left Command Check

              breq    USART_LFT

              cpi           cpr, 0b11110001             ; Halt Command Check

              breq    USART_HLT

              cpi           cpr, 0b11110010             ; Freeze Command Check

              breq    USART_FRZ


              rjmp    MAIN                        ;  Jump  back  to  main  in  order  to  create  a
loop


USART_FWD:

              LDI           mpr, 0b00000001             ; Use the PORTB lights to check for
routine call

              OUT           portB, mpr


              rcall   USART_ADDR                  ; Subroutine to send the address first


Check_FWD:

              lds           spr, UCSR1A                 ;  Check  for  the  empty  flag  from
control reg A
```

```
            sbrs    spr, UDRE1                  ; Check Specific bit
            rjmp    Check_FWD                   ; Repeat until empty
            ldi             mpr, MovFwd                 ;   Prepare   to   send   mov   forward
command
            sts             UDR1, mpr                   ; Place into UDR for Tx
            rjmp    MAIN                        ; Jump back to main loop



USART_BCK:
            LDI             mpr, 0b00000010             ;   Use  PORTB  lights  to  check  for
subroutine call
            OUT             portB, mpr
            rcall   USART_ADDR                  ; Call to send the address


Check_BCK:
            lds             spr, UCSR1A                 ;  Check  to  see  if  the  register  is
empty for transmission
            sbrs    spr, UDRE1                  ; using the UDRE flag
            rjmp    Check_BCK
            ldi             mpr, MovBck                 ; Prepare move back command
            sts             UDR1, mpr                   ; Send
            rjmp    MAIN                        ; Go back to main


USART_RGT:
            LDI             mpr, 0b00000100             ;  use  PORTB  lights  to  check  for
subroutine call
            OUT             portB, mpr


            rcall   USART_ADDR


Check_RGT:
```

```
            lds         spr, UCSR1A                 ; Check that UDR is ready using the
empty flag in A
            sbrs    spr, UDRE1
            rjmp    Check_RGT                   ; loop till objective met
            ldi         mpr, TurnR                  ;    prepare    turn    right    for
transmission
            sts         UDR1, mpr                   ; Transmission send
            rjmp    MAIN                        ; Jump back to main

USART_LFT:
            LDI         mpr, 0b00001000             ;  Use  PORTB  lights  to  check  for
subroutine call
            OUT         portB, mpr

            rcall   USART_ADDR                  ; Send the address first

Check_LFT:
            lds         spr, UCSR1A                 ; Check that UDRE is set so we know
that we can Tx
            sbrs    spr, UDRE1
            rjmp    Check_LFT                   ; Keep looping till its empty
            ldi         mpr, TurnL                  ; Load Turn Left
            sts         UDR1, mpr                   ; Send it
            rjmp    MAIN                        ; Return to main loop

USART_HLT:
            LDI         mpr, 0b00010000             ;  Use  PORTB  lights  to  check  for
subroutine call
            OUT         portB, mpr


            rcall   USART_ADDR                  ; Send the address first
```

```
Check_HLT:
                lds             spr, UCSR1A                     ; Check to see that the UDRE flag is
set so that we know to transmit
                sbrs    spr, UDRE1
                rjmp    Check_HLT                       ; Loop until condition is met
                ldi             mpr, Halt                       ; Load the halt command
                sts             UDR1, mpr                       ; Send it
                rjmp    MAIN                            ; Go back to the main loop


USART_FRZ:
                LDI             mpr, 0b10000000                 ;   Use   PORTB   to   show   this   is
happening
                OUT             portB, mpr



                rcall   USART_ADDR                      ; Send that address over


Check_FRZ:
                lds             spr, UCSR1A                     ; Wait until the UDRE flag is set so
we can send
                sbrs    spr, UDRE1
                rjmp    Check_FRZ                       ; Keep waiting till conditions
                ldi             mpr, Freeze                     ; Prepare freeze command
                sts             UDR1, mpr                       ; Send Freeze
                rjmp    MAIN                            ; Jump to main program


USART_ADDR:
                lds             spr, UCSR1A                     ; Wait until the UDRE flag is set so
we can send
                sbrs    spr, UDRE1
                rjmp    USART_ADDR                      ; kEEP WAITING TILL READY
```

```
                ldi             mpr, $2F                        ; Send the address first
                sts             UDR1, mpr
                ret



;***********************************************************
;*      Functions and Subroutines
;***********************************************************


;***********************************************************
;*      Stored Program Data
;***********************************************************


;***********************************************************
;*      Additional Program Includes
;***********************************************************
```

Source Code - Receiver

```
;***********************************************************
;*
;*      Faaiq_Waqar_and_Jordan_Brown_lab8_Rx_sourcecode.asm
;*
;*      This is the USART reciever
;*
;*      This is the RECEIVE skeleton file for Lab 8 of ECE 375
;*
;***********************************************************
;*
;*       Author: Faaiq Waqar & Jordan Brown
```

```
;*         Date: December 5th 2019
;*
;***********************************************************

.include "m128def.inc"                    ; Include definition file

;***********************************************************
;*     Internal Register Definitions and Constants
;***********************************************************
.def   o2lcnt = r24
.def   i2lcnt = r23
.def   ilcnt = r22
.def   olcnt = r21
.def   waitcnt = r20
.def   dpr = r19                          ; DeadBot-Purpose Register
.def   cpr = r18                          ; Comparison-Purpose Register
.def   spr = r17                          ; Secondary-Purpose Register
.def   mpr = r16                          ; Multi-Purpose Register

.equ   WskrR = 0                          ; Right Whisker Input Bit
.equ   WskrL = 1                          ; Left Whisker Input Bit
.equ   EngEnR = 4                         ; Right Engine Enable Bit
.equ   EngEnL = 7                         ; Left Engine Enable Bit
.equ   EngDirR = 5                        ; Right Engine Direction Bit
.equ   EngDirL = 6                        ; Left Engine Direction Bit

.equ   BotAddress = $2F;(Enter your robot's address here (8 bits))

;/////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
```

```
;***********************************************************

.equ    MovFwd =  (1<<EngDirR|1<<EngDirL)    ;0b01100000 Move Forward Action Code

.equ    MovBck =  $00                                    ;0b00000000   Move   Backward   Action
Code

.equ    TurnR =   (1<<EngDirL)                           ;0b01000000 Turn Right Action Code

.equ    TurnL =   (1<<EngDirR)                           ;0b00100000 Turn Left Action Code

.equ    Halt =    (1<<EngEnR|1<<EngEnL)            ;0b10010000 Halt Action Code

.equ    Freeze =  0b01010101


.equ    MovFwdCmd =  ($80|1<<(EngDirR-1)|1<<(EngDirL-1))   ;0b10110000 Move Forward Action Code

.equ    MovBckCmd =  ($80|$00)
;0b10000000 Move Backward Action Code

.equ    TurnRCmd =   ($80|1<<(EngDirL-1))                                ;0b10100000       Turn
Right Action Code

.equ    TurnLCmd =   ($80|1<<(EngDirR-1))                                ;0b10010000  Turn  Left
Action Code

.equ    HaltCmd =    ($80|1<<(EngEnR-1)|1<<(EngEnL-1))            ;0b11001000 Halt Action Code

.equ    FreezeCmd =  0b11111000


;***********************************************************

;*      Start of Code Segment

;***********************************************************

.cseg                                            ; Beginning of code segment


;***********************************************************

;*      Interrupt Vectors

;***********************************************************

.org    $0000                            ; Beginning of IVs

              rcall   INIT                ; Reset interrupt

.org    $0002

              rcall   BUMP_RIGHT

              reti
```

```
.org    $0004
                rcall   BUMP_LEFT
                reti
.org    $003C
                rcall   USART_FUNC
                reti

;Should have Interrupt vectors for:
;- Left whisker
;- Right whisker
;- USART receive

.org    $0046                           ; End of Interrupt Vectors

;***********************************************************
;*      Program Initialization
;***********************************************************
INIT:
        ;Stack Pointer (VERY IMPORTANT!!!!)
        ldi             mpr, high(RAMEND)
        out             SPH, mpr
        ldi             mpr, low(RAMEND)
        out             SPL, mpr
        ;I/O Ports
        ldi             mpr, $FF
        out             DDRB, mpr
        ldi             mpr, $00
        out             PORTB, mpr

        ldi             mpr, (0<<0|0<<1|0<<2|0<<3|1<<4|0<<5|0<<6|0<<7)
```

```
out             DDRD, mpr
ldi             mpr, (1<<0|1<<1)
out             PORTD, mpr
;USART1
ldi             mpr, high(832) ; Do We Round up or Down?
sts             UBRR1H, mpr
ldi             mpr, low(832) ; Same Question applies here
sts             UBRR1L, mpr
        ;Enable transmitter
ldi             mpr, (1<<U2X1)
sts             UCSR1A, mpr
ldi             mpr, (1<<RXEN1|0<<TXEN1|1<<RXCIE1)
sts             UCSR1B, mpr
ldi             mpr, (0<<UPM11|1<<USBS1|1<<UCSZ11|1<<UCSZ10)
sts             UCSR1C, mpr
;External Interrupts
ldi             mpr, (1<<ISC01|0<<ISC00|1<<ISC11|0<<ISC20||1<<ISC21)
sts             EICRA, mpr
ldi             mpr, (1<<INT0|1<<INT1)
out             EIMSK, mpr
        ;Set the External Interrupt Mask
        ;Set the Interrupt Sense Control to falling edge detection


;Other
;Set Timer Ready for Wait Usage


ldi             cpr, MovFwd
ldi             dpr, MovBck
ldi             waitcnt, 100
```

```
sei


;***********************************************************
;*      Main Program
;***********************************************************
MAIN:
        ;TODO: ???
        out             PORTB, cpr
        rjmp    MAIN


DEAD_MAIN:
        ldi             cpr, Halt
        out             PORTB, cpr
        rjmp    DEAD_MAIN


KILL_SWITCH:
        ldi             mpr, (0<<TXC1|0<<U2X1|0<<MPCM1) ; Set Everything to 0
        sts             UCSR1A, mpr                                     ; Send to control reg
A
        ldi             mpr, (0<<RXEN1|0<<TXEN1|0<<RXCIE1|0<<UCSZ11)       ; Continue to do this
for the rest
        sts             UCSR1B, mpr
        ldi             mpr,
(0<<UMSEL1|0<<UPM11|0<<UPM10|0<<USBS1|0<<UCSZ11|0<<UCSZ10|0<<UCPOL1)
        ;External Interrupts
        ldi             mpr, (0<<ISC01|0<<ISC00|0<<ISC11|0<<ISC10)
        sts             EICRA, mpr
        ldi             mpr, (0<<INT0|0<<INT1)
        out             EIMSK, mpr                                      ;
        rjmp DEAD_MAIN
```

```
;***********************************************************
;*      Functions and Subroutines
;***********************************************************
BUMP_RIGHT:
                push    mpr                     ; Save mpr register
                in              mpr, SREG       ; Save program state
                push    mpr                     ;


                ldi             mpr, (0<<TXEN1|0<<UCSZ12)
                sts             UCSR1B, mpr


                ; Move Backwards for a second
                ldi             mpr, MovBck     ; Load Move Backward command
                out             PORTB, mpr      ; Send command to port
                rcall   Wait                    ; Call wait function


                ; Turn left for a second
                ldi             mpr, TurnL      ; Load Turn Left Command
                out             PORTB, mpr      ; Send command to port
                rcall   Wait                    ; Call wait function


                ; Move Forward again
                ldi             mpr, MovFwd     ; Load Move Forward command
                out             PORTB, mpr      ; Send command to port


                ldi             mpr, (1<<RXEN1|0<<TXEN1|1<<RXCIE1)
                sts             UCSR1B, mpr


                pop             mpr             ; Restore program state
                out             SREG, mpr       ;
```

```
                pop             mpr             ; Restore mpr


                ldi             mpr,$FF
                out             EIFR, mpr


                ret                             ; Return from subroutine


BUMP_LEFT:
                push    mpr                     ; Save mpr register
                in              mpr, SREG       ; Save program state
                push    mpr                     ;


                ldi             mpr, (0<<TXEN1|0<<UCSZ12)
                sts             UCSR1B, mpr


                ; Move Backwards for a second
                ldi             mpr, MovBck     ; Load Move Backward command
                out             PORTB, mpr      ; Send command to port
                rcall   Wait                    ; Call wait function


                ; Turn right for a second
                ldi             mpr, TurnR      ; Load Turn Left Command
                out             PORTB, mpr      ; Send command to port
                rcall   Wait                    ; Call wait function


                ; Move Forward again
                ldi             mpr, MovFwd     ; Load Move Forward command
                out             PORTB, mpr      ; Send command to port


                ldi             mpr, (1<<RXEN1|0<<TXEN1|1<<RXCIE1)
```

```
                sts             UCSR1B, mpr

                pop             mpr             ; Restore program state
                out             SREG, mpr       ;
                pop             mpr             ; Restore mpr

                ldi             mpr,$FF
                out             EIFR, mpr

                ret                             ; Return from subroutine

Wait:
                push    waitcnt                 ; Save wait register
                push    ilcnt                   ; Save ilcnt register
                push    olcnt                   ; Save olcnt register

Loop:   ldi             olcnt, 224              ; load olcnt register
OLoop:  ldi             ilcnt, 237              ; load ilcnt register
ILoop:  dec             ilcnt                   ; decrement ilcnt
                brne    ILoop                   ; Continue Inner Loop
                dec             olcnt                   ; decrement olcnt
                brne    OLoop                   ; Continue Outer Loop
                dec             waitcnt                 ; Decrement wait
                brne    Loop                    ; Continue Wait loop

                pop             olcnt           ; Restore olcnt register
                pop             ilcnt           ; Restore ilcnt register
                pop             waitcnt         ; Restore wait register
                ret
```

```
USART_FUNC:
                lds             mpr, UDR1               ; Load Contents from Reciever
                cpi             mpr, Freeze             ; Compare to the freeze
                breq    FROZEN                  ; Freeze the robot
                cpi             mpr, $2F                ; Compare to robot address
                breq    USART_CONT              ; If equivalent, check for the command
                brne    RETURN_U_BAD    ; Finih program if wrong address


USART_CONT:
                lds             mpr, UDR1               ; Load new contents
                cpi             mpr, MovFwdCmd ; Check for MovFwd Command
                breq    COM_FWD
                cpi             mpr, MovBckCmd ; Check for MovBck Command
                breq    COM_BCK
                cpi             mpr, TurnRCmd  ; Check for TurnR Command
                breq    COM_RGT
                cpi             mpr, TurnLCmd  ; Check for TurnL Command
                breq    COM_LFT
                cpi             mpr, HaltCmd   ; Check for Halt Command
                breq    COM_HLT
                cpi             mpr, FreezeCmd ; Check for Freeze Command
                breq    COM_FRZ


RETURN_U_BAD:
                ldi             mpr,$FF                 ; Make sure to eliminate Queued interupts
                out             EIFR, mpr               ; Clear by setting to the flag register
                ret


COM_FWD:
                ldi             cpr, MovFwd             ; Load Forward Instruction to cpr
```

```
                rjmp    RETURN_U_BAD
COM_BCK:
                ldi             cpr, MovBck             ; Load Backwards Instruction to cpr
                rjmp    RETURN_U_BAD
COM_RGT:
                ldi             cpr, TurnR              ; Load Turn Right to cpr
                rjmp    RETURN_U_BAD
COM_LFT:
                ldi             cpr, TurnL              ; Load Turn Left to cpr
                rjmp    RETURN_U_BAD
COM_HLT:
                ldi             cpr, Halt               ; Load Halt to cpr
                rjmp    RETURN_U_BAD
COM_FRZ:
                ldi             mpr, (1<<TXEN1|0<<UCSZ12|0<<RXEN1|0<<RXCIE1) ; Disable recieve
                sts             UCSR1B, mpr
        ; Use UCSR1B to make Tx


                lds             spr, UCSR1A             ; Wait until the bit is free
                sbrs    spr, UDRE1
                rjmp    COM_FRZ
                ldi             mpr, Freeze             ; Send over to the Data Reg
                sts             UDR1, mpr
                rcall   Wait                    ; Wait so we dont freeze ourself


                ldi             mpr, (1<<RXEN1|0<<TXEN1|1<<RXCIE1)   ; Set back to normal
                sts             UCSR1B, mpr                                             ; Place
in UCSR1B


                rjmp    RETURN_U_BAD   ; Return
```

```
FROZEN:
            ldi         mpr, (0<<INT0|0<<INT1)      ; Disable Bumpbot
            out         EIMSK, mpr

            inc         dpr                                 ; Increment number of
fatalities
            cpi         dpr, 3
            breq    DEAD_JUMP                       ; Go finish bot if 3 freezes
            ldi         mpr, Halt                           ; Use halt
            out         PORTB, mpr                          ; Place to PORTB
            rcall   WAIT_5                          ; Wait for 5 sec

            ldi         mpr, (1<<INT0|1<<INT1)      ; Renable Bumpbot
            out         EIMSK, mpr

            rjmp    RETURN_U_BAD

DEAD_JUMP:
            rjmp    KILL_SWITCH         ; Go to the end of robot life

Wait_5:
            push    waitcnt             ; Save wait register
            push    ilcnt               ; Save ilcnt register
            push    olcnt               ; Save olcnt register

            ldi         i2lcnt, 2
            ldi         olcnt, 150          ; load olcnt register
            ldi         ilcnt, 216          ; load ilcnt register
            ldi         o2lcnt, 9
Loop_5:
```

```
                dec             o2lcnt                  ; decrement ilcnt
                brne    Loop_5                  ; Continue Inner Loop
                dec             ilcnt                   ; decrement ilcnt
                brne    Loop_5                  ; Continue Inner Loop
                dec             olcnt                   ; decrement olcnt
                brne    Loop_5                  ; Continue Outer Loop
                dec             i2lcnt                  ; Decrement wait
                brne    Loop_5                  ; Continue Wait loop

                pop             olcnt           ; Restore olcnt register
                pop             ilcnt           ; Restore ilcnt register
                pop             waitcnt         ; Restore wait register
                ret
;************************************************************
;*      Stored Program Data
;************************************************************

;************************************************************
;*      Additional Program Includes
;************************************************************
```